再帰プログラムの意味論について*

長谷川 真人

1 はじめに

計算とはなにか，という問いに対して，今のところ，誰もが合意するような直接的，一般的かつ数学的な回答は知られていない．計算の理論においても，あるいは現実のコンピュータ（電子計算機のみならず電卓でも算盤でも構わないが）においても，我々は，計算を，なんらかの表現を介して理解し，分析し，利用しているのであって，計算そのものについての普遍的な定式化があるわけではないのである．将来そのような定式化が与えられる可能性がないわけではない．しかし，現時点では，我々の計算という現象へのアプローチは，すべて計算を表現する手段に深く依存したものである．従って，計算について語るためには，計算の表現についても語ることが不可欠である．

だから，現在の計算機科学は，不可避的に ‘ 計算の表現論 ’ とでもいうべき性格を持っている．計算とはなにか，という問い方をするのではなく，計算を，どのような構造のうえでどのように表現できるか，そしてその表現からいかに有用な結果を導くことができるか，さらに異なる表現の間にいかなる関係が成り立つかというのが，計算機科学者の本質的な（しかしたいてい暗黙の）問題意識なのである．それが端的にあらわれているのが，プログラミング言語の理論，特にこれから述べる**プログラミング言語の意味論（semantics of programming languages）**である．

プログラミング言語は，計算の持つ構造をコンピュータ上に表現する，非常に強力な手段である．たいていの実用的なプログラミング言語は，豊富なデータ構造と制御構造をそなえている．データ構造は計算の対象の構造を表現し，また制御構造は計算の進め方の構造を明らかにする．その他にも，プログラミング言語の多くの機能が，計算の表現に明快な構造を与えるために存在している．プログラミング言語を用いてプログラムを書くことが，すでに，計算に ‘ 良い表現 ’ を与える作業であるとさえ考えられる．しかし，話はここで終わらない．プログラミングを通じてなんらかの構造を与えられた計算（の表現）に対して，我々は，それらの構造の性格を調べることで，実に多くのことを知ることができる．プログラムの構造の持つ性質に着目し，適切に抽象化された数学的な対象を用いて表現することによって，そのプログラムがあらわす計算の意味することをシステマティックに解読する，それがプログラミング言語の意味論の基本的な目標である．

現在のプログラミング言語の意味論は，現存する（実用的なものだけでも数千以上といわれる）多彩なプログラミング言語群からも推察できるように，プログラミング言語の実に多様な機能に即して発展しつつあり，その全貌について語るのは，本稿の目的ではない．以下では，プログラミング言語の制御構造の中でももっとも基本的な**再帰（recursion）**について焦点をあてて論じる．あ

*数学 **59** (2007), 180–191

れこれ欲張るより，ひとつの中心的なテーマを固定して述べるほうが，結果的にこの分野の特質をより明確にできると考えたのが主な理由であるが，同時に，筆者自身のこれまでの研究の多くが再帰プログラムの意味論に関わるものであったということも理由のひとつである．まず，古典的な再帰プログラムの不動点意味論の基礎について，概説する．そのあと，より最近の研究の展開について，筆者の理解の及ぶ範囲で紹介する．

2 再帰プログラムの不動点意味論

2.1 再帰から不動点へ

プログラムのなかで自分自身を呼び出すことを再帰呼び出しという．再帰呼び出しを行うプログラムのことを再帰的に定義されたプログラム，または単に再帰プログラムと呼ぶ．たとえば，整数を入力に取り整数を出力するプログラム

$$\mathsf{fact}(x) \ \equiv\ \mathsf{if}\ x = 0\ \mathsf{then}\ 1\ \mathsf{else}\ x \times \mathsf{fact}(x-1)$$

は，再帰プログラムの典型的な例である．再帰プログラムは，一般には，常に停止するとは限らない（関数として well-defined であるとは限らない）ので，その働きは，数学的には関数ではなく，部分関数として理解する必要がある．実際，上のプログラムは，非負整数の入力についてはその階乗を出力するが，負の整数に対しては，いつまでも答えを出力しない，すなわち未定義であることが容易に想像できる．(より現実的なプログラミングでは，プログラムは単に部分関数を表現するにとどまらず，コンピュータの状態を変化させる，いわゆる ‘ 副作用 ’ を伴うことが一般的である．本稿では，問題を単純化するため，特に断らない限り，副作用を起こさない，いわゆる ‘ 関数型プログラム ’ についてだけ考える.)

このような再帰プログラムが実際にどのような部分関数を定めるのかを知るのは，決して自明な問題ではない．そもそも，プログラムと入力値から，その実行結果が定義されているかどうか判定すること（停止性問題）は決定不可能，つまり機械的に遂行可能な決定方法が存在しないことがわかっている．しかし，ここでは，決定可能性・計算可能性の問題には立ち入らないで，まずは，再帰プログラムの定める部分関数を数学的にきちんと確定させることを考えよう．

そのための基礎となるアイデアは，再帰プログラム（のあらわす部分関数）を，適切な（汎）関数の不動点として理解する，というものである．$\mathbf{Z} \rightharpoonup \mathbf{Z}$ を，整数上の部分関数全体の集合とする．以下のように，$\mathbf{Z} \rightharpoonup \mathbf{Z}$ 上の（汎）関数 $F : (\mathbf{Z} \rightharpoonup \mathbf{Z}) \to (\mathbf{Z} \rightharpoonup \mathbf{Z})$ を考える．

$$F(f)(x) \ =\ \begin{cases} 1 & (x = 0) \\ x \times f(x-1) & (x \neq 0,\ f(x-1)\ \text{が定義されている場合}) \\ \text{未定義} & (\text{その他の場合}). \end{cases}$$

ここで，当然のことながら，F の定義には再帰は用いられていないことに注意されたい．また，F は通常の意味での（全域的な）関数である．さて，さきほどの再帰プログラム fact を考えよう．fact があらわしている部分関数を $[\![\mathsf{fact}]\!] : \mathbf{Z} \rightharpoonup \mathbf{Z}$ と書くことにする．具体的には，

$$[\![\mathsf{fact}]\!](x) \ =\ \begin{cases} x! & (x \geq 0) \\ \text{未定義} & (\text{その他の場合}) \end{cases}$$

となるはずである．ここで，F の定義のなかの f に $[\![\mathsf{fact}]\!]$ を代入すると

$$F([\![\mathsf{fact}]\!])(x) = \begin{cases} 1 & (x=0) \\ x \times y & (x \neq 0,\ [\![\mathsf{fact}]\!](x-1)=y) \\ \text{未定義} & (\text{その他の場合}) \end{cases} = \begin{cases} x! & (x \geq 0) \\ \text{未定義} & (\text{その他の場合}) \end{cases} = [\![\mathsf{fact}]\!](x),$$

したがって，$[\![\mathsf{fact}]\!]$ は（部分関数としての）等式 $F([\![\mathsf{fact}]\!]) = [\![\mathsf{fact}]\!]$ を満たす．言い換えると，$[\![\mathsf{fact}]\!]$ は，F の不動点である．

以上の議論はまったく自明なものであるが，再帰プログラムの意味が，適切な数学構造上の不動点として定式化可能であるという，大変重要な指針を示唆するものである．ただし，それだけでは，再帰プログラムの定める部分関数を確定させるという問題はまだ解決していない．実は F の不動点は一意には定まらないのだが，その中で，$[\![\mathsf{fact}]\!]$ をどう特徴づけすればよいか，という問題が残っている．その問題に明快な解答を与える古典的な理論が，これから紹介する領域理論である．

2.2 領域理論

1960 年代末に生まれた**領域理論（domain theory）**では，データの集まり（データ型）をデータ領域あるいは単に領域とよばれる適切な順序構造によって，またデータを処理するプログラムを領域の間の（適切な意味で）連続な関数として捉える（[16]）．複雑な離散的な構造に関する問題を，連続的なたちの良い構造に帰着させて考える，好例といえよう．領域理論の有効性は，以下で述べる再帰プログラムの不動点意味論のみならず，再帰データ型とよばれる複雑な構造に対する領域の明快な構成を与えることによって，プログラミング言語の意味論の標準的な理論のひとつとして発展してきた．領域として用いる順序構造には様々な選択肢があるが，以下では，もっとも単純な，完備半順序集合による定式化を与える．

半順序集合 $D = (D, \sqsubseteq)$ は，最小元をもち，また，任意の可算単調列 $d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq \ldots$ の上限 $\bigsqcup d_n$ が存在するとき，**完備半順序集合（complete partial order, CPO）**（より正確には最小元を持つ ω-完備半順序集合）と呼ばれる．D が CPO であるとき，その最小元を $\bot_D$ または単に $\bot$ であらわす．

CPO D, E について，関数 $f : D \to E$ は，以下の条件をみたすとき**連続（continuous）**であるという．

- 単調性：$d \sqsubseteq d'$ ならば $f(d) \sqsubseteq f(d')$
- 連続性：任意の単調列 $d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq \ldots$ について $f(\bigsqcup d_n) = \bigsqcup f(d_n)$

（以下の議論には用いないが，CPO に適当な位相を入れることにより，上に与えた CPO 間の連続関数の定義を，この位相についての連続関数のそれとして理解することもできる．CPO D の部分集合 U は，以下の条件を満たすとき Scott 開集合であるという：

1. $x \in U$ かつ $x \sqsubseteq y$ ならば $y \in U$
2. $\{x_0 \sqsubseteq x_1 \sqsubseteq \ldots\} \subseteq D$ かつ $\bigsqcup x_i \in U$ ならば，ある i について $x_i \in U$

この定義により，D は位相空間とみなせる．上に与えた CPO 間の連続関数の定義は，この位相についての連続関数のそれと一致する.)

以上の準備のもとで，以下の最小不動点定理が得られる．

定理 1. *CPO D と連続関数 $f : D \to D$ について，$f(d) = d$ を満たす $d \in D$ で最小のものが存在し，それは $\bigsqcup f^n(\bot)$ で与えられる．*

証明は平易である：f の連続性から $f(\bigsqcup f^n(\bot)) = \bigsqcup f^{n+1}(\bot) = \bigsqcup f^n(\bot)$ が成り立つ．また，$f(d) = d$ とすると，$\bot \sqsubseteq d$ より $f^n(\bot) \sqsubseteq f^n(d) = d$ なので，$\bigsqcup f^n(\bot) \sqsubseteq d$ である．

例として，先に考えた，整数上の部分関数全体 $\mathbf{Z} \rightharpoonup \mathbf{Z}$ を思い出そう．これは，以下の順序 $\sqsubseteq$ に関して CPO をなす：

$$f \sqsubseteq g \iff \text{任意の } x \in \mathbf{Z} \text{ について } f(x) \text{ が定義されているならば } g(x) \text{ も定義されていてしかも } f(x) = g(x)$$

最小元 $\bot$ は，任意の $x \in \mathbf{Z}$ について対応する値が未定義であるような部分関数である．また，$f_0 \sqsubseteq f_1 \sqsubseteq f_2 \sqsubseteq \ldots$ について，

$$(\bigsqcup f_i)(x) = \begin{cases} f_k(x) & (\text{ある } k \text{ について } f_k(x) \text{ が定義されているとき}) \\ \text{未定義} & (\text{その他の場合}) \end{cases}$$

この完備汎順序に対して，さきほど考えた汎関数 F は，連続になる．そして，$F^n(\bot) : \mathbf{Z} \rightharpoonup \mathbf{Z}$ は

$$F^n(\bot)(x) = \begin{cases} x! & (0 \le x < n) \\ \text{未定義} & (\text{その他の場合}) \end{cases}$$

である．したがって，F の最小不動点 $\bigsqcup_n F^n(\bot)$ は $[\![\mathsf{fact}]\!]$ に他ならない．実際，$F^n(\bot)$ は，fact の実行を高々n 回の再帰呼び出しまでで打ち切った場合を表現しているので，その上限が fact の実行の正しい表現を与えることは，自然なことである．もちろん，その他にも F の不動点は存在する（非負整数では階乗を，負の数では 0 を返す関数など）．しかし，それらは，もともとの fact には規定されていない余計な計算を行うものであり，fact の定義から構成された F の最小不動点 $[\![\mathsf{fact}]\!]$ こそ，fact の表現する部分関数として適切なものである．

以上では，領域理論のごく初歩的な事柄しか使っていない．しかし，この，再帰プログラムを最小不動点によって特徴付けるという発想は，プログラミング言語の意味論のもっとも重要な洞察のひとつである．再帰データ型に対応する領域の構成という，より困難な問題も，実は最小不動点の構成を拡張することによって非常にエレガントに解くことができる．再帰プログラムがプログラムの上の演算の不動点として捉えられるのに対し，再帰データ型は，プログラムの集まりの上の演算の不動点として捉えることができる．一般には後者の構成のほうが困難であるが，実は，適切な条件のもとでは，これらのふたつの問題は同値となる．領域理論は，そのような条件を満たす構造の研究として理解することもできる（[6]，[5]）．ここでは，そういった領域理論のより顕著な部分およびその応用については解説しないが，興味を持たれた読者は，是非適切な文献を参照されたい（[2]，[7]，[20]）．

3　巡回構造と再帰プログラム

前節では，再帰プログラムの領域理論を用いた（最小）不動点意味論という，古典的なアプローチについて概説した．この不動点意味論は簡潔かつ明快な理論であり，すでに述べたように，プログラミング言語の意味論において中心的な役割を果たしている．

けれども，不動点意味論は，再帰プログラムに表現される計算が持っている重要な側面を過度に単純化しているきらいがある．どういうことかというと，不動点意味論は，たしかに再帰プログラムの数学的な解釈を定めてくれるけれども，その一方，どのように再帰的な計算が生み出されるかについては，ほとんど情報をもたらさない． 最小不動点の構成が，実際のコンピュータ上での再帰プログラムの実装と，そのまま対応しているわけではないのである． また，これに関連して，再帰と，プログラミング言語の他の機能との相互作用についてもうまく説明できない，という問題もある．

そこで，このような，領域理論では捉えられない問題に対して，適切に抽象化された情報を与えてくれるような数学構造を，必ずしも従来の不動点意味論には拘らずに，特定する必要が生じてくる．以下では，この十年ほどの間になされた，この方向での理論展開について解説する．

3.1　巡回構造から生み出される再帰

領域理論の誕生から遅れること数年，1970 年代半ばに，再帰プログラムを，巡回的なグラフ構造として解釈し，コンピュータ上で実現する方法が考案された．図 1 に示すように，再帰プログラムを巡回的なグラフとして表現し，その実行を，グラフの書き換えとして実現する，というのが，基本的なアイデアである．これは，早くから関数型プログラミング言語の効率的な実装方法として用いられ（[18]），後にグラフ書き換えに基づく計算の理論として定式化された．

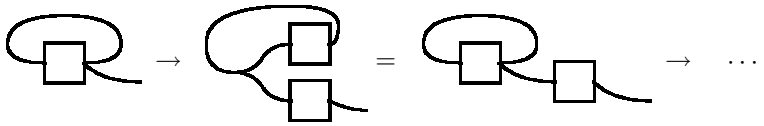


図 1: 再帰呼び出し = 巡回的に共有された資源のコピー

図 1 では非常に単純な場合を考えたが，プログラミング言語の他の機能と組み合わせることで，再帰的な計算の表現および実行も，より複雑なものになってくる．たとえば，巡回的ラムダ計算 [3] と呼ばれる，関数型プログラミング言語に明示的に巡回構造を付加した状況を説明できる計算モデルでは，再帰を生み出すプログラム（再帰演算子）を，図 2 のように，様々な異なる方法で定義することができる．これらは，それぞれ異なったふるまいをする（図 3）．

今，これらのプログラム（あるいは巡回グラフ）は ‘ 異なる ’ と書いた．それは，視覚的にはそうである．しかし，数学的には，これらが異なるということをどのように理解したらいいのだろうか? これは，全く自明でない問題である．実は，領域理論と最小不動点を用いて解釈すると，これらは同一の連続関数に対応してしまうのである．実際，適切な条件のもとでは，プログラマの視点では，これらは同一の働きをする（同一の結果を出す）といえるので，領域理論の結果は妥当なものではある．けれども，コンピュータ上のふるまいを分析すると，これらにはやはり違いがある，と考えたくなる． 簡単な例として，整数の入力 x に対し，0 または $x+1$ を非決定的に返すプログラム P

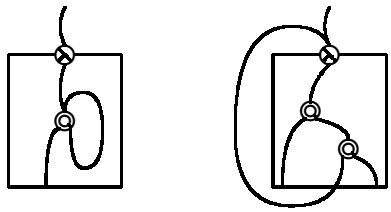


図 2: 巡回的ラムダ計算における再帰演算子のふたつの実装

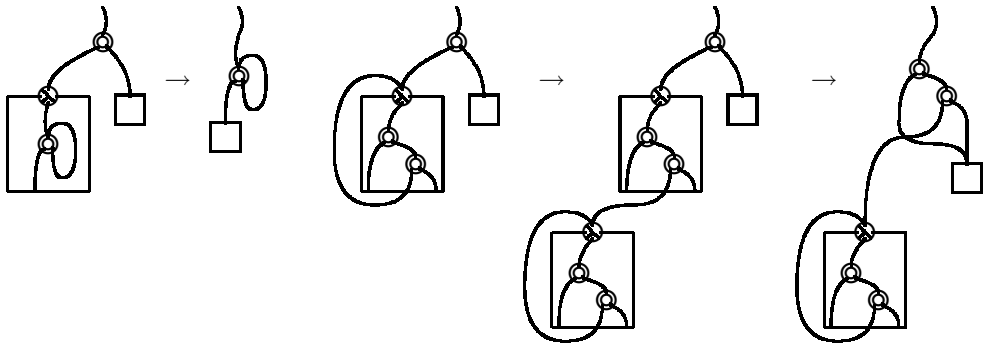


図 3: 再帰演算子のふるまいの例

を考える．P の ‘ 不動点 ’ として得られるプログラムの出力には，非決定性の起きるタイミングにより，以下のような可能性が考えられる：

(A) $P(x) = 0$ または $x + 1$ である．前者の場合，その不動点は 0 である．後者の場合，不動点は存在せず，これは永久にとまらない計算 $((\cdots + 1) + 1) + 1$ をあらわす．したがって，期待できる答えは 0 または未定義となる．

(B) $P(x) = 0$ または $x + 1$ であり，前者の場合，その不動点は 0 である．後者の場合，x に $P(x)$ を当てはめて，$P(x) + 1$ を計算する．もしも $P(x) = 0$ ならば，答えは $0 + 1 = 1$ である．そうでなければ，x に $P(x)$ を当てはめて，$(P(x) + 1) + 1$ を計算する．以下同様に続けていけば，可能な答えは 0, 1, 2,... または未定義である．

実は，これらのふたつの可能性は，図 2 のふたつの再帰演算子をこの P に適用した結果に，それぞれ対応している（小さい正方形に P をあてはめる）．非決定性が一回で解消されるか，あるいは再帰のたびに引き起こされるかで，このような違いが出てくるのである．

この例に限らず，与えられたふたつのプログラムが同じ働きをするだろうか，という問題は，プログラミング言語の理論と応用において，極めて基本的な問いかけである．一般には非常に困難な（決定不可能な）問題であるが，ある程度の制約のもとではかなり現実的な答えを与えることも可能である．領域理論も役に立つ — もしふたつのプログラムが異なる連続関数に対応していたら，それらははっきり異なるのである．けれども，この例のように，同一の連続関数に対応している場合，今考えているようなデリケートな差異は説明できない．まして，この例のような，そのままでは関

数としてはとらえられないようなプログラムを考慮に入れたい場合、プログラムを（部分）関数とみなす領域理論の枠組みが不適切であることは容易に想像できる．

ここで，領域理論からひとまず離れることにする．領域理論でなくとも，プログラムの実行に関して，不変な，抽象的な性質を考察できる枠組みさえあれば，プログラミング言語の意味論は展開できるはずである．今考えているような例を説明できるような，そんな‘ 計算の不変量 ’がほしい — そのためのアイデアと技法は，このような巡回構造を扱うことに長けた数学の一分野の中に見出すことができる．結び目と，その不変量の理論 [15] である．

ここで述べるまでもないことだが，結び目の不変量の理論は，たとえば，図 4 のふたつの結び目図が同じ結び目をあらわすだろうか，という問題について，有用な指針を与えてくれる．結び目の不変量では，結び目の連続的な変形に対して不変な数学的な量（たとえば多項式）を考察する．もしもこれらの図に異なる量を対応させるような不変量があれば，これらが異なる結び目をあらわすことがわかる．

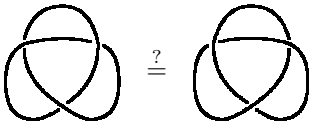


図 4: 結び目の分類問題

この，結び目の不変量の考え方は，上述したプログラミング言語の意味論のそれと，大変よく似ている．実際，これは，ただの比喩ではない．実は，結び目の不変量のために考え出された数学構造を，巡回構造から生まれる再帰計算の意味論に，そのまま用いることができる．結び目の不変量と，再帰計算の意味論には，共通する数学構造が存在するのである．

3.2 トレース付きモノイダル圏

1980 年代以降，結び目の不変量の研究は大きく様変わりした．特に，量子不変量の登場から，結び目の不変量の代数的・圏論的な扱いが劇的に進んだ（[14]，[19]）．ここで扱うのは，その一例である，トレース付きモノイダル圏 [13] の構造である．

ところで，計算機科学の話題に圏論が登場するのは，今では珍しいことではない．プログラミング言語の持つ複雑な構造について語るとき，圏論はもっとも有用な数学的な枠組みのひとつである．領域理論にしても，圏論の言葉でその本質的な特徴を明確に捉えることができる．以下で述べることも，プログラミング言語の意味論の伝統から，(そっくりそのまま収まるわけではないにしても) 大きく逸脱するものではないことを，強調しておく．

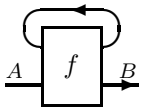
モノイダル圏（monoidal category） $\mathbf{C} = (\mathbf{C}, \otimes, I, a, l, r)$ とは, 圏 $\mathbf{C}$ と, それに付随するテンソル積またはモノイダル積と呼ばれる関手 $\otimes : \mathbf{C}^2 \to \mathbf{C}$, 単位対象と呼ばれる $\mathbf{C}$ の対象 I, テンソル積と単位対象の結合律・単位律に相当し適当な公理をみたす可逆な自然変換 $a_{A,B,C} : (A \otimes B) \otimes C \xrightarrow{\sim} A \otimes (B \otimes C)$, $l_A : I \otimes A \xrightarrow{\sim} A$ および $r_A : A \otimes I \xrightarrow{\sim} A$ の組のことをいう．モノイダル圏で, $a_{B,C,A} \circ c_{A,B\otimes C} \circ a_{A,B,C} = (1_B \otimes c_{A,C}) \circ a_{B,A,C} \circ (c_{A,B} \otimes 1_C)$ （双線型性）と $c^{-1}_{A,B} = c_{B,A}$（対称性）を満たす可逆な自然変換 $c_{A,B} : A \otimes B \xrightarrow{\sim} B \otimes A$ を持つものを, **対称モノイダル圏（symmetric monoidal category）**と呼ぶ. また, この定義から対称性を除き, かわりに c^{-1} の双線型性を仮定し

たより一般的な構造が**ブレイド付きモノイダル圏（braided monoidal category）**（c はブレイドと呼ばれる）である．さらに，バランスあるいはツイストとよばれる（リボンや枠付きタングルのねじれに対応する）自然変換 $\theta_A : A \xrightarrow{\sim} A$ が存在して $\theta_I = 1_I$ および $\theta_{A\otimes B} = c_{B,A} \circ (\theta_B \otimes \theta_A) \circ c_{A,B}$ をみたすとき，**バランス付きモノイダル圏（balanced monoidal category）**という．対称モノイダル圏は，バランス付きモノイダル圏の特殊な（$\theta_A = 1_A$ であるような）場合である．以上の概念の詳細については [12]，[17]，[19] を参照されたい．ブレイド付きモノイダル圏，バランス付きモノイダル圏は，絡み目の不変量の議論において本質的である．一方，再帰プログラムの意味論では，$c_{A,B}$ と $c^{-1}_{B,A}$ を区別する必要がないので，対称モノイダル圏のみ考えれば十分である．

いま，バランス付きモノイダル圏 $\mathbf{C}$ が与えられているものとする．このとき，$\mathbf{C}$ 上の**トレース** [13] とは，$\mathbf{C}$ の対象で添字付けられた関数の族

$$Tr^X_{A,B} : \mathbf{C}(A \otimes X, B \otimes X) \to \mathbf{C}(A, B)$$

で，図 5 に示す条件を満たすものである．理解を容易にするために，$\mathbf{C}$ の射のグラフィカルな表現を併記する．たとえば，$f : A \otimes X \to B \otimes X$ のトレース $Tr^X_{A,B}(f) : A \to B$ を



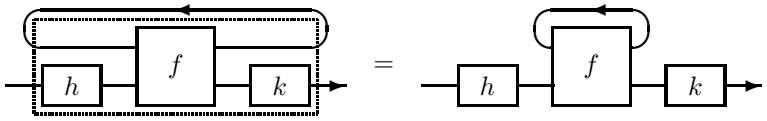
と表現することにする．また，$c_{X,Y}$ を交差  で，$c^{-1}_{X,Y}$ を  で，θ_X をねじれ  で，さらに θ^{-1}_X を逆のねじれ  で表現する．なお，簡単のため，最後のふたつの条件式において結合律 a は省略されている（実際，strict なモノイダル圏のみを考えても一般性は失われない）．

トレースを持つバランス付きモノイダル圏を，**トレース付きモノイダル圏（traced monoidal category）**と呼ぶ．なお，図 5 で与えた条件は，もとの定義 [13] のものを若干簡単にしたものである．古典的な例としては，体 K 上の有限次元線型空間と線型写像のなす圏は，トレース付き（対称）モノイダル圏になっている．トレースは行列の対角和（の一般化）である：線型写像 $f : U \otimes_K W \to V \otimes_K W$ について，そのトレース $Tr^W_{U,V}(f) : U \to V$ は，$(Tr^W_{U,V}(f))_{i,j} = \Sigma_k f_{i\otimes k, j\otimes k}$ で与えられる．

より最近の重要な例として，量子群の表現全体のなす圏があげられる．これは，自明でない（対称的でない）ブレイドを持つ例であり，この圏で結び目図を解釈することで，いわゆる量子不変量が得られる（[14]）．たとえば，三つ葉結び目は図 6 のようにトレース（ブレイド閉包）を用いて $Tr^X_{X,X}(c_{X,X} \circ c_{X,X} \circ c_{X,X})$ と解釈できる．

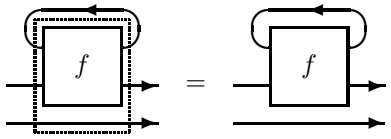
結び目の理論では，リボン圏（ribbon categories）またはトーティルモノイダル圏（tortile monoidal categories）などと呼ばれる，適切な条件を満たす双対を持つ（しばしば autonomous，compact または rigid などといわれる）バランス付きモノイダル圏を考えることが多い（[14]，[17]，[19]）．量子群の表現の圏は，リボン圏の重要な例である．結び目の不変量との関係では，1 点集合から自由生成されるリボン圏が（向きづけられた）フレーム付きタングルの圏とモノイダル同値であることが本質的である [17]．ところで，よく知られているように，リボン圏ではトレースを一意に定めることができる．逆に，任意のトレース付きモノイダル圏から，それを充満かつ忠実に埋め込めるようなリボン圏が構成でき，しかもすべてのリボン圏はそのようにして得られるものと同値である（[13]）．すなわち，双対を持つという性質を忘れて，トレースだけに着目しても，本質的な情報は失われない．特に，再帰プログラムの意味論では，一般に双対を持つことは仮定できないので，双対を用いないトレースによる定式化は本質的である．

$$Tr^X_{A',B'}((k \otimes 1_X) \circ f \circ (h \otimes 1_X)) = k \circ Tr^X_{A,B}(f) \circ h$$



$$Tr^X_{X,X}(c_{X,X}) \circ \theta_X^{-1} = 1_X = Tr^X_{X,X}(c^{-1}_{X,X}) \circ \theta_X$$

$$Tr^X_{C\otimes A, C\otimes B}(1_C \otimes f) = 1_C \otimes Tr^X_{A,B}(f)$$



$$Tr^X_{A,B}(Tr^Y_{A\otimes X, B\otimes X}(f)) = Tr^Y_{A,B}(Tr^X_{A\otimes Y, B\otimes Y}((1_B \otimes c_{Y,X}) \circ f \circ (1_A \otimes c^{-1}_{Y,X})))$$

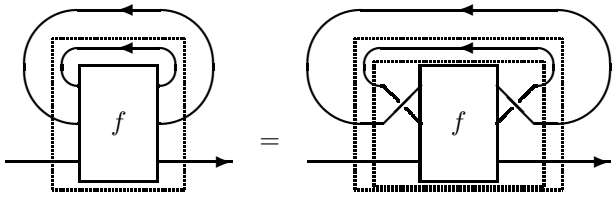


図 5: トレースの公理

3.3 再帰プログラムの‘不変量’

上の結び目の場合と同様の発想で，巡回構造を用いた再帰プログラムもまたトレース付きモノイダル圏で解釈できる（不変量を与えることができる）.

まず，領域理論による不動点意味論が，実はトレース付きモノイダル圏の特殊な場合になっていることを説明する．領域理論では，CPO の直積をテンソル積とみることで，CPO と連続関数の圏は対称モノイダル圏となる．そのような，テンソル積が直積であるような状況では，再帰的プログラムの解釈に用いる不動点演算子が存在することと，トレースが存在することがまったく同値であることが，Martin Hyland と筆者によって独立に証明された（[8]）．直積を持つ圏 $\mathbf{C}$ において、以下の条件をみたす関数の族 $(-)^\dagger : \mathbf{C}(A \times X, X) \to \mathbf{C}(A, X)$ を Conway 不動点演算子と呼ぶ（[4]）：

- $f : A \times X \to X$, $h : A' \to A$ について $f^\dagger \circ h = (f \circ (h \times 1_X))^\dagger : A' \to X$
- $f : A \times Y \to X$, $g : A \times X \to Y$ について $(f \circ \langle \pi_{A,X}, g \rangle)^\dagger = f \circ \langle 1_A, (g \circ \langle \pi_{A,Y}, f \rangle)^\dagger \rangle : A \to X$

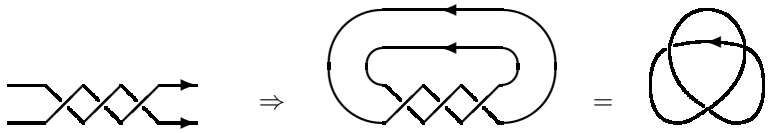


図 6: 三つ葉結び目の構成

- $f : A \times X \times X \to X$ について $f^{\dagger\dagger} = (f \circ (1_A \times \Delta_X))^{\dagger} : A \to X$

これらの条件は，従来の再帰プログラムの不動点意味論のほとんどすべてについて成立している．たとえば、CPO と連続関数の圏では，$f^{\dagger}(a)$ を連続関数 $x \mapsto f(a, x)$ の最小不動点とすることで Conway 不動点演算子が定まる．

定理 2. 有限直積を持つ圏 $\mathbf{C}$ においては, 以下の条件は同値である．

1. $\mathbf{C}$ は（直積をテンソル積とみなした対称モノイダル圏の構造に対して）トレースを持つ．
2. $\mathbf{C}$ は *Conway* 不動点演算子を持つ．

実際，トレースから Conway 不動点演算子が，$f^{\dagger} = Tr^X_{A,X}(\Delta_X \circ f)$（ただし $f : A \times X \to X$）として構成できる．パラメータ A に関する部分を無視すれば，これは図 1 の巡回共有グラフから再帰を作り出す構成そのものである．また，$f = \langle f_0, f_1 \rangle : A \times X \to B \times X$（ただし $f_0 : A \times X \to B$, $f_1 : A \times X \to X$）について，そのトレースは $Tr^X_{A,B} f = f_0 \circ \langle 1_A, f_1^{\dagger} \rangle$ として与えられる．このトレースと Conway 不動点演算子との対応は，適切な一様性を保つことがわかっており（[10]），非常に自然なものである．

とくに，領域理論の知られているすべてのモデルはトレースを持つ．このことは，最小不動点が Conway 不動点演算子を与えることからわかる．実際，領域理論の多くのモデルでは，Conway 不動点演算子はただひとつしか存在しないことが知られており，そのような場合には，トレースは一意に定まる．また, 最小不動点によらないモデル（たとえば長谷川立による組み合わせ論的な不動点 [11]）であっても，多くの場合にトレースが再帰的プログラムの解釈を特徴づけていることがわかっている．したがって，古典的な表示的意味論の枠組みでは，不動点意味論を与えることはトレースの構造を与えることとほぼ同義といってよい．

その一方で，巡回的ラムダ計算のようなより複雑な場合について適切なモデルを与えるには，テンソル積が必ずしも直積ではないような状況に，表示的意味論を一般化した場合を考える必要がある．そのような場合についても，適当な条件のもとで，トレースの存在が再帰プログラムの意味論を与えることがわかる：

定理 3. 有限直積を持つ圏から, トレース付き対称モノイダル圏への対称モノイダル左随伴関手が存在するとき, このトレース付きモノイダル圏は *dinatural* な不動点演算子を持つ．

術語の詳細，証明，およびこのような数学構造を用いたグラフ書き換え系の意味論とその具体例については，文献 [8]，[9] を参照されたい．こうして得られた新しい再帰プログラムのモデルでは，従来の不動点意味論よりも精密に再帰的プログラムの分類を行うことができる．いわば，より精度の高い不変量が得られるわけである．たとえば，図 2 のふたつの再帰演算子は，非決定性計算のモデル

から構成される‘不変量’によって，明確に区別される．この意味論では，非決定性関数 $f : X \to X$（関数 $f : X \to 2^X$）について，前者は集合 $\{x \in X \mid x \in f(x)\}$ を、また後者は $f(A) = A$ となるような X の部分集合 A で最大のものを対応させる．前に，非決定的にふるまうプログラムを用いてこれらの再帰演算子が異なる動作をする簡単な例を与えたが，実はその例はこの意味論に即して導いたものである．

以上で述べた最近の理論展開の根底にあるのは，再帰プログラムの意味論が，結び目の不変量の理論と共通した構造を持つという大きな見通しであり，そしてそれを裏付けるトレース付きモノイダル圏の構造である．この，トレース付きモノイダル圏による再帰プログラムの意味論の一般化は，今日までに，多くの新しい再帰計算のモデルの発見・構成をもたらしている．

また，再帰プログラムの意味論との関わりとは若干異なるが深く関連した話題として，前に触れたトレース付きモノイダル圏からリボン圏を構成する方法が，相互作用的（双方向的）な計算のモデルを構成する技法に対応することが知られており（[1]），この方向でも活発な研究が続けられている．

このように，トレース付きモノイダル圏は，理論計算機科学における重要な基本的な構造のひとつとして認知されてきた．この事例にとどまらず，計算という現象の本質に迫る良い表現・良い構造が，実は数学の諸分野においても重要な構造である可能性は，極めて大きいと思う．そのような，数学と計算機科学に共通する構造の発見と応用が，今後も，様々な状況でなされていくと期待している．

文献

[1] S. Abramsky, Retracting some paths in process algebra, In: Proc. Concurrency Theory, Lecture Notes in Comput. Sci., **1119** (1996), 1–17.

[2] S. Abramsky and A. Jung, Domain theory, In: Handbook of Logic in Computer Science Vol. 3, Oxford Univ. Press (1994), 1–168.

[3] Z.M. Ariola and J.W. Klop, Cyclic lambda graph rewriting, In: Proc. Logic in Computer Science, IEEE Press (1994), 416–425.

[4] S. Bloom and Z. Ésik, Iteration Theories, EATCS Monographs on Theoretical Computer Science, Springer, 1993.

[5] M.P. Fiore, Axiomatic Domain Theory in Categories of Partial Maps, Distinguished Dissertations in Computer Science, Cambridge Univ. Press, 1996.

[6] P. Freyd, Algebraically compact categories, In: Category Theory, Lecture Notes in Math., **1488** (1991), 95–104.

[7] C.A. Gunter, Semantics of Programming Languages, MIT Press, 1992.

[8] M. Hasegawa, Recursion from cyclic sharing: traced monoidal categories and models of cyclic lambda calculi, In: Proc. Typed Lambda Calculi and Applications, Lecture Notes in Comput. Sci., **1210** (1997), 196–213.

[9] M. Hasegawa, Models of Sharing Graphs: A Categorical Semantics of let and letrec, Distinguished Dissertation Series, Springer, 1999.

[10] M. Hasegawa, The uniformity principle on traced monoidal categories, Publ. Res. Inst. Math. Sci., **40** (2004), 991–1014.

[11] R. Hasegawa, Two applications of analytic functors, Theoret. Comput. Sci., **272** (2002), 113–175.

[12] A. Joyal and R. Street, Braided tensor categories, Adv. Math., **102** (1993), 20–78.

[13] A. Joyal, R. Street and D. Verity, Traced monoidal categories, Math. Proc. Cambridge Philos. Soc., **119** (1996), 447–468.

[14] C. Kassel, Quantum Groups, Grad. Texts in Math. **155**, Springer, 1995.

[15] W.B.R. Lickorish, An Introduction to Knot Theory, Grad. Texts in Math., **175**, Springer, 1997.

[16] D. Scott, A type theoretical alternative to CUCH, ISWIM, OWHY, privately circulated (1969); published in Theoret. Comput. Sci., **121** (1993), 233–247.

[17] M.-C. Shum, Tortile tensor categories, J. Pure Appl. Algebra, **93** (1994), 57–110.

[18] D.A. Turner, A new implementation technique for applicative languages, Software – Practice and Experience, **9** (1979), 31–49.

[19] D.N. Yetter, Functorial Knot Theory, Ser. Knots Everything, **26**, World Scientific, 2001.

[20] 横内寛文, プログラム意味論, 情報数学講座, **7**, 共立出版, 1994.

(2006 年 7 月 19 日提出)

(はせがわ まさひと・京都大学数理解析研究所)